VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Analýza faktur za pomoci obrazu

Invoice analysis using computer vision

2018 Miloslav Szczypka

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Miloslav Szczypka**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Analýza faktur za pomocí obrazů
Invoice Analysis Using Computer Vision

Jazyk vypracování: čeština

Zásady pro vypracování:

V posledních letech díky rozvoji algoritmů z oblasti počítačového vidění a strojového učení těší velké oblibě i optické rozpoznání znaků a digitalizace tištěných textů. Touto problematikou se také zabývá tato práce.

1. Seznamte s existujícími OCR algoritmy.
2. Jeden z přístupů vyberte a s jeho pomocí vytvořte aplikaci pro analýzu faktur.
3. Zaměřte se na čtení základních informací o faktuře (například číslo faktury, celkovou částku, datum splatnosti).
4. Experimentálně otestujte funkčnost navrženého řešení.
5. V závěru práce se zamyslete nad budoucím vylepšením (například rozšíření funkčnosti o čtení položek na fakturách).

Seznam doporučené odborné literatury:

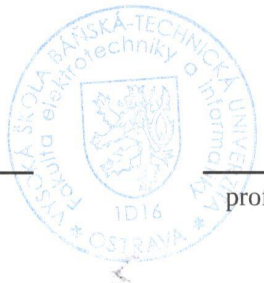
[1] Babenko, B., Belongie, S.J., & Wang, K. End-to-end scene text recognition, International Conference on Computer Vision, 1457-1464, 2011.
[2] Max Jaderberg, Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. 2016. Reading Text in the Wild with Convolutional Neural Networks. Int. J. Comput. Vision 116, 1 (January 2016), 1-20. DOI: http://dx.doi.org/10.1007/s11263-015-0823-z

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Radovan Fusek, Ph.D.**

Datum zadání: 01.09.2017

Datum odevzdání: 30.04.2018



doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 30. dubna 2018 .....

Rád bych poděkoval Radovanu Fuskovi, vedoucímu mé práce, za pomoc při řešení této bakalřské práce.

Abstrakt

V posledních letech díky rozvoji algoritmů z oblasti počítačového vidění a strojového učení se těší velké oblibě i optické rozpoznání znaků a digitalizace tištěných textů. Touto problematikou se také zabývá tato práce.

Cílem této práce je vytvoření programu, který automaticky detekuje základní informace z faktury a umožní s nimi dále pracovat (např. export do různých formátů).

Klíčová slova: Zpracování obrazu, strojové vidění, OCR, faktury, optické rozpoznání znaků, hluboké učení

Abstract

Recently optical character recognition and digitalization of printed text is becoming more popular thanks to development of computer vision algorithms. This work is also focused on this topic.

Goal of this work is to create a program, which automatically detects basic invoice information and allows the user to proces them furthermore (for example export to various formats).

Key Words: Image processing, computer vision, OCR, Invoices, optical character recognition, deeplearning

Obsah

Seznam použitých zkratek a symbolů

PPM	–	Portable Pixel Map
PNM	–	Portable Any Map
JPEG	–	Joint Photographic Experts Group
PNG	–	Portable Network Graphics
GNU	–	GNU's Not Unix
LSTM	–	Long Short Term Memory
UTF	–	Unicode Transformation Format
HTML	–	Hypertext Markup Language
dpi	–	dots per inch

Seznam obrázků

Seznam tabulek

1 Úvod

Moderní technologie přináší nové možnosti automatizace v různých oblastech. Ne jinak tomu je i v oblasti strojového vidění. Strojové vidění v dnešní době může přinést mnoho zjednodušení a urychlení práce. Oblast využití je široká, využívá se například u asistenčních systému nových automobilů, ve výrobě při kontrole nebo třízení výrobků, ale také při zpracování dokumentů a mnoho dalších.

Využití strojového vidění v oblasti zpracování dokumentů je díky zvyšování výkonu a pokroků v oblasti hlubokého učení stále větší. Můžeme zpracovávat a převádět tištěné nebo ručně psané dokumenty do digitální formy a dále s nimi pracovat.

Vstup z tištěných dokumentů není vždy ideální, proto je nutné ještě před samotným zpracováním textu provést předzpracování, které zvýší úspěšnost rozeznaní textu. Pro předzpracování dokumentu tato práce využívá knihovnu OpenCV.

K převodu tištěných dokumentů do digitální podoby se využívá různých Optical Character Recognition (OCR) algoritmů, tedy v češtině optické rozpoznávaní znaků. Díky těmto algoritmům je možné převádět velké množství tištěného textu do digitální podoby, např. skenování knih.

Tato práce využívá předzpracování, OCR algoritmus a následné zpracování a vyhledání klíčových informací. Výstup může být v různých standardních formátech, aby se dal využít v různých systémech, které s daty dále pracují.

Některé informace v testovacích a ilustračních obrázcích v této práci byly cenzurovány z důvodu výskytu osobních informací jako například telefonní čísla nebo čísla bankovních účtů.

2 Předzpracování pro OCR Algoritmy

Téměř všechny OCR algoritmy vyžadují určité předzpracování obrazu pro dosažení nejlepších výsledků. Prvním krokem při práci s OCR algoritmy je tudíž vylepšení kvality vstupního obrazu. Pokud se tento krok neprovede, některé OCR algoritmy v sobě mají zabudovány mechanizmy předzpracování obrazu, avšak ani tehdy výsledek nemusí být optimální. Nejčastějšími metodami předzpracování jsou převod barevného obrazu na černobílý, odfiltrování šumu různými způsoby, zostření obrazu nebo binarizace. V této kapitole budou popsány metody předzpracování, které se používají snad u všech aplikací v oboru zpracování obrazu.

2.1 Převod na stupně šedi

Převod na stupně šedi se provádí, protože zde není nutnost pracovat s celým barevným spektrem, tudíž místo toho, aby měl každý pixel 3 složky RGB (red, green, blue), bude mít pouze složku jednu a to jas. Díky tomu se sníží počet dat a lépe se pracuje s černobílým obrazem, jelikož má jen jednu barevnou složku. V mém řešení převádím obraz na stupně šedi kvůli toho, že rozlišovat barvy není třeba a zbytečně bych musel pracovat se třemi složkami barev. K převodu se využívá tento vzorec:

$$GrayScale = 0.299R + 0.587G + 0.114B \tag{1}$$

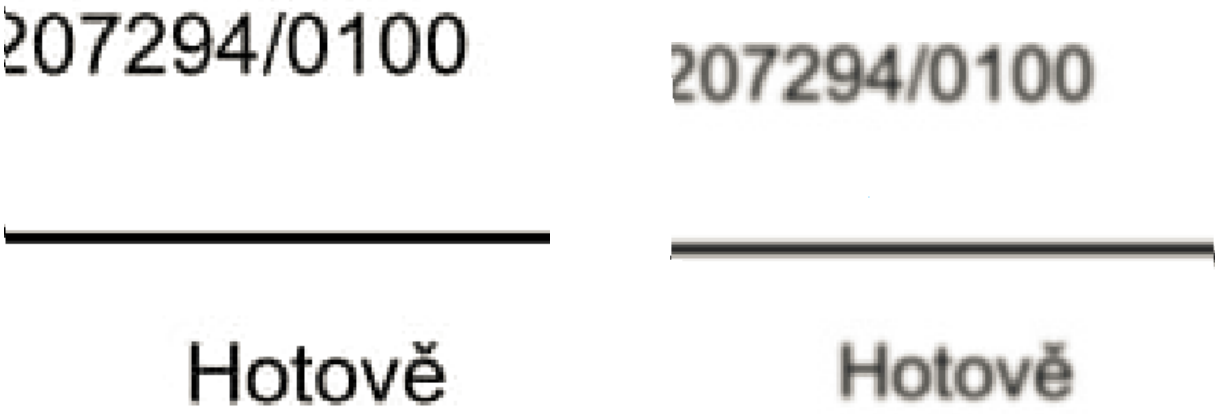
2.2 Filtrování obrazu

Převádění tištěných dokumentů do elektronické podoby se nazývá digitalizace. Při digitalizaci vzniká různé množství nežádoucího šumu v závislosti na použité technologii. Je tedy zřejmé, že i takový šum by mohl negativně ovlivnit čtení dokumentu, tudíž se aplikují různé metody jak tento šum odstranit či alespoň částečně potlačit.

Gaussův filtr

Jedním z možných řešení jak potlačit šum je Gaussův filtr. Tento filtr není tak agresivní, jako ostatní metody. Přesný vzorec Gaussovy funkce je:

$$G(x,y) = \frac{1}{\sqrt{2\pi}\sigma} \cdot e^{\frac{-x^2+y^2}{2\sigma^2}} \tag{2}$$



(a) Před použitím Gaussova filtru (b) Po aplikování Gaussova filtru

Obrázek 1: Příklad použit Gaussova filtru

Kvůli rychlosti výpočtu se však nepoužívá tento vzorec, ale používá se matice, která se blíží hodnotě tohoto vzorce. Výsledná matice se pak musí podělit součtem všech elementů v matici. Příklad takového výpočtu je:

$$G = \frac{1}{273}\begin{vmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{vmatrix} \tag{3}$$

Lze si všimnout, že hodnoty v matici opravu odpovídají tvaru Gaussovy křivky. Hodnoty uprostřed jsou nejvyšší a matice je kruhová. Gaussův filtr v této práci používám při odstranění šumu při vyrovnávání faktury a při odstraňování rámečků.

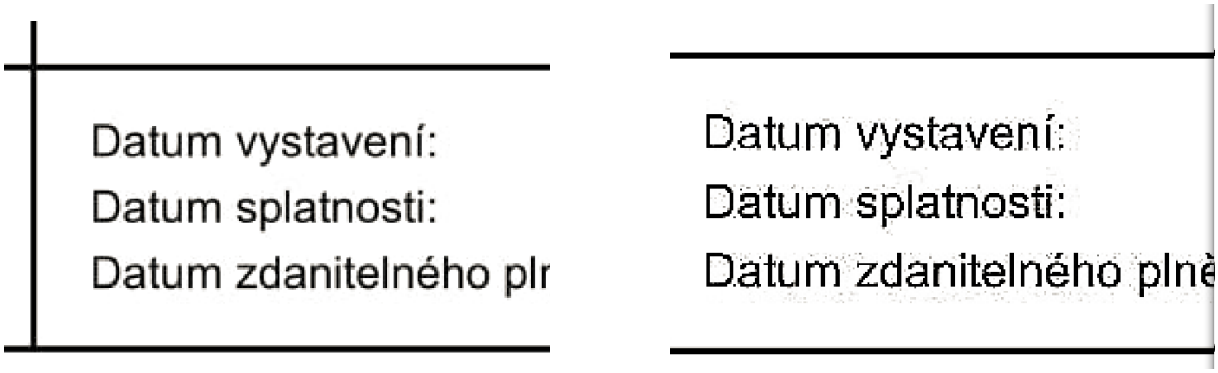
Existují i další metody jak potlačit šum. Jedna z možností je například prostý průměr z okolních pixelů, nebo jejich medián. Já však v této práci používám Gaussův filtr. Příklad použití Gaussova filtru je na obrázku 1.

2.3 Zostření obrazu

Výsledkem nedokonalé digitalizace nebo špatně vytištěné faktury je rozmazaný nebo rozostřený obraz. Aby bylo možné ho úspěšně přečíst, lze provést jeho zostření, které toto rozmazání eliminuje a výsledkem bude ostřejší obraz s ostřejšími hranami.

Konvoluce

Mezi používané metody zostření obrazu patří použití konvoluce se správně zvolenou maticí. Jedná se v podstatě o stejný proces jako u rozmazání obrazu, kdy se vytvoří filtr, který je



(a) Před použitím konvoluce (b) Po aplikování konvoluce

Obrázek 2: Příklad použit konvoluce

posouván po obrazu. Rozdíl je však v hodnotách, které v tomto filtru jsou. Matici, která se používá pro zostření obrazu se říká Laplacián. Běžné verze Laplaciánu vypadají následovně. Každá z těchto matic má různou intenzitu zostření obrazu.

$$\begin{vmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{vmatrix} \quad nebo \quad \begin{vmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{vmatrix} \tag{4}$$

Na obrázku 2 byla použita první matice a je jasně zřetelné, že hrany jsou výraznější, avšak množství šumu, které díky zostření vzniklo citelně vzrostlo. Tento šum by mohl negativně ovlivnit čtení dokumentu OCR algoritmem.

Odečtení rozostřeného obrazu

Jiná možnost jak zostřit obraz je v angličtině uváděna jako "unsharp masking". Jedná se o postup, kdy je původní obraz rozostřen Gaussovým filtrem a následně je "odečten"od původního obrazu, což zařídí doostření obrazu. Tento postup nemá tak výrazný výsledek jako při použití Laplaciánu.

Obrázek 3 ilustruje použití této metody. Jde také poznat, že tato metoda tolik nezvyšuje množství šumu ve zostřeném obrazu a i přesto jsou hranové přechody zřetelnější.

2.4 Cannyho detektor

Před detekcí natočení dokumentu se provádí detekce hran, jejíž výsledkem jsou hranové body (pixely), pomocí kterých je možno detekovat přímky.

(a) Původní dokument

(b) Dokument na který byl aplikován Gaussův filtr

(c) Vyostřený dokument

Obrázek 3: Ukázka zostření pomocí odečtení rozmazaného obrazu

Jedním z mnoha detektorů hran je Cannyho detektor. Není to však jen jednoduchý detektor hran jako například Sobelův detektor, který se skládá pouze ze dvou matic. Cannyho detektor se skládá z více kroků, díky čemuž je poměrně přesný.

Na obrázku 4 je uvedeno praktické použití Cannyho detektoru. Není těžké si všimnout, že Cannyho detektor detekoval dvě hrany pro každé písmeno/čáru v dokumentu. V mnoha aplikacích je tento efekt nechtěný, ovšem při mém použití to není takový problém, jelikož v mé práci používám Cannyho detektor k nalezení vodorovných přímek, pomocí kterých je možné poznat úhel, pod kterým je dokument natočen.

Potlačení šumu a výpočet velikosti a směru gradientu

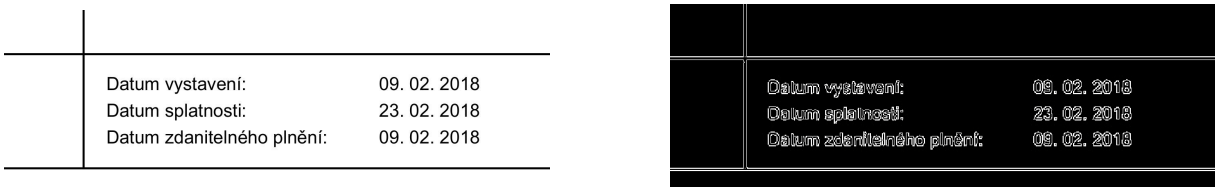
Nejprve se obraz zbaví šumu. V tomto kroku lze použít například Gaussův filtr. V další fázi se zjišťují gradienty, které určují, zda je v konkrétním místě hrana nebo ne a jakým směrem je hrana orientována. K tomuto se používá například Sobelův operátor.

Určení bodů, které jsou lokálními maximy

Po určení gradientu je oblast hrany poměrně velká. Proto se hledají lokální maxima po a proti směru, kterým je gradient orientován. Po nalezení tohoto maxima zůstane pouze bod s největší intenzitou gradientu jako hrana.

Prahování s hysterezí

V posledním kroku se podle intenzity gradientu rozhoduje co je hrana a co ne. Pro tento krok



(a) Původní obraz

(b) Výsledek Cannyho detektoru

Obrázek 4: Ukázka použití Cannyho detektoru

je nutné si zvolit dva prahy. Jeden minimální a druhý maximální. Pokud je hodnota daného pixelu nižší než minimální práh, není to hrana. Pokud je však jeho hodnota větší než maximální práh, pak je to určitě hrana. Jestliže se intenzita bodu nachází mezi těmito prahy, tak se tento bod bere za hranu pouze tehdy, pokud je propojen s jiným bodem, který je nad maximálním prahem.

2.5 Detekce natočení a vyrovnání dokumentu

Problém, který často nastává při digitalizaci je to, že dokument není umístěn přesně a tím pádem může být dokument různě nakloněný. Většina OCR algoritmů se sama o narovnání nedokáže postarat, takže je to většinou krok zahrnutý v procesu předzpracování obrazu, jelikož takto nakloněné dokumenty pak nejsou správně OCR algoritmem přečteny. Jeden z možných přístupů jak detekovat a vyrovnat otočení dokumentu, je za použití Houghovy transformace.

Houghova transformace

Aby bylo možné detekovat otočení dokumentu, tak se musí nejprve zjistit pod jakým úhlem je dokument natočen. K tomuto se používá právě Houghova transformace, která dokáže detekovat jak pozici přímky, tak i její úhel natočení.

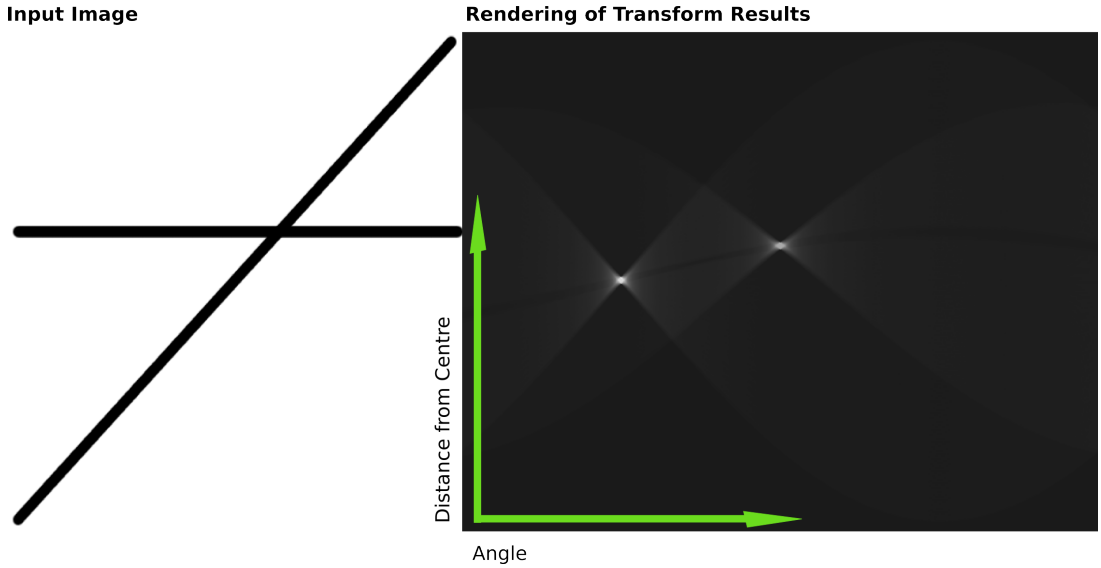
Klasická rovnice přímky vypadá následovně:

$$y = ax + b \tag{5}$$

Pomocí této rovnice je možno přímku zakreslit do roviny x,y. Pokud je však tato rovnice upravena do tvaru:

$$b = -ax + y \tag{6}$$

Je možné jednotlivé hranové body reprezentovat přímkou v rovině a,b. Pak lze zjistit, zda body leží na stejné přímce a to tak, že se protnou v jednom bodě, který odpovídá jejich parametrům a a b. Problém u tohoto přístupu však nastává ve chvíli, kdy je přímka rovnoběžná s osou Y.



Obrázek 5: Detekce přímek pomocí Houghovy transformace[1]

Řešením tohoto problému je přechod do polárního souřadného systému. Rovnice přímky v polárním souřadném systému je:

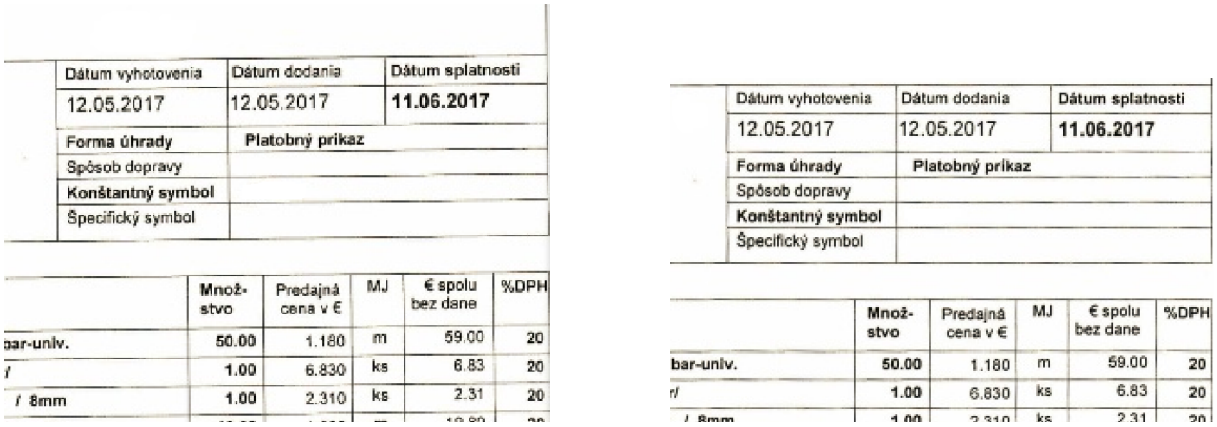
$$\rho = x \cdot \cos\theta + y \cdot \sin\theta \tag{7}$$

kde ρ je vzdálenost přímky od počátku soustavy a θ je úhel přímky od počátku.

Algoritmický postup pro detekci přímek je následující:

1. Výpočet velikosti diagonály obrazu
2. Určení množiny ρmin až ρmax v intervalu <-diagonála, +diagonála>
3. Určení množiny θmin až θmax v intervalu $<-90°, 90°>$
4. Vytvoření 2D pole o velikosti početρ x početθ
5. Pro všechny hranové body se provede:
    6. Pro všechny prvky množiny ρmin až ρmax se provede:
        7. Výpočet vzorce polární rovnice přímky uvedený výše pro konkrétní hodnotu ρ
        8. Prvek 2D pole na pozici [ρ, θ] se zvýší o 1
9. Zvolení minimálního počtu bodů ležících na přímce
10. Vyhledání všechny hodnot v 2D poli větší než je minimum, což odpovídá detekovaným přímkám

Obrázek 5 ilustruje toto řešení. Z obrázku lze vidět, že Houghova transformace detekovala 2 přímky podle dvou výrazných bodů na obrazu.



Dátum vyhotovenia	Dátum dodania	Dátum splatnosti
12.05.2017	12.05.2017	11.06.2017
Forma úhrady	Platobný príkaz	
Spôsob dopravy		
Konštantný symbol		
Špecifický symbol		

	Množstvo	Predajná cena v €	MJ	€ spolu bez dane	%DPH
bar-univ.	50.00	1.180	m	59.00	20
r/	1.00	6.830	ks	6.83	20
r/ 8mm	1.00	2.310	ks	2.31	20

(a) Původní křivě naskenovaný dokument

Dátum vyhotovenia	Dátum dodania	Dátum splatnosti
12.05.2017	12.05.2017	11.06.2017
Forma úhrady	Platobný príkaz	
Spôsob dopravy		
Konštantný symbol		
Špecifický symbol		

	Množstvo	Predajná cena v €	MJ	€ spolu bez dane	%DPH
bar-univ.	50.00	1.180	m	59.00	20
r/	1.00	6.830	ks	6.83	20

(b) Opravený dokument

Obrázek 6: Ukázka korekce natočení dokumentu

Detekce a oprava natočení

Princip samotné detekce a opravy natočení dokumentu je následující. Nejdříve se pomocí Houghovy transformace detekují vodorovné přímky přičemž samotná detekce zjistí jejich úhel natočení. Poté se zprůměruje úhel všech přímek a vytvoří se rotační matice. Tato matice se pak aplikuje na obraz čímž se dokument narovná. Publikace [12] pojednává o jiné, robustnější metodě detekce a opravení natočení.

Na obrázku 6 lze vidět srovnání původního, křivě naskenovaného a vyrovnaného dokumentu tímto algoritmem.

2.6 Morfologické operace

Morfologické operace jsou operace při zpracování obrazu, které slouží k filtrování obrazu. Mezi atomické operace se řadí dilatace a eroze. Tyto operace se provádí nejčastěji nad binárním obrazem, ale lze je provést i nad černobílým nebo i barevným obrazem.

Morfologické operace potřebují dva prvky. Obraz který se bude filtrovat a strukturní element, který slouží jako filtr. Tento filtr se poté aplikuje na celý obraz stejně jako třeba v případě gaussova filtru. Morfologické filtry mohou mít různé tvary. Může to být čtverec, kruh, elipsa nebo i kříž. Pro lepší představivost zde uvedu i obrázky konkrétního použití jednotlivých operací.

Dilatace

Dilatace je proces, který zvětšuje objekty na popředí binárního obrazu. Pokud je alespoň jedna hodnota pod filtrem rovna jedničce, tak je celá oblast po filtrem rovna jedničce. Tato operace je znázorněna na obrázku 7.

(a) Před dilatací (b) Po dilataci

Obrázek 7: Ukázka dilatace

(a) Před erozí (b) Po erozi

Obrázek 8: Ukázka eroze

Eroze

Eroze je opačný proces oproti dilataci. Výsledná oblast se rovná jedničce pokud jsou všechny hodnoty pod filtrem rovny jedné. Výsledkem této operace je tedy ztenčený objekt. Příklad použití eroze je na obrázku 8.

Otevírání

Otevírání je proces eroze následovaný dilatací. Tento proces slouží k filtrování šumu. Filtruje bílý šum na pozadí binárního obrazu. Nejúčinnější je proti šumu "salt-and-pepper". Ukázka operace otevírání je na obrázku 9.

Zavírání

Zavírání je opět proces opačný k otevírání. Je to dilatace následovaná erozí. Tato technika filtruje černý šum, který se vyskytuje v objektu. Proces zavírání ilustruje obrázek 10.

Další operace

Existuje mnoho dalších morfologických operací jako například morfologický gradient. Tyto ostatní operace jsou však stále tvořeny základními dvěma morfologickými operacemi (dilatací a erozí).

(a) Před otevíráním (b) Po otevírání

Obrázek 9: Ukázka otevírání

(a) Před zavíráním (b) Po zavírání

Obrázek 10: Ukázka zavírání

Rozdíly pak jsou buď ve tvaru strukturního elementu nebo v následné práci s výsledky eroze a dilatace.

2.7 Binarizace

Další krokem v předzpracování je binarizace obrazu. Tento postup převede černobílý obraz pouze na jedničky a nuly, kdy jednička reprezentuje bílou barvu a nula zase černou barvu nebo naopak. V takovém výsledném obraze se pak vytvoří tzv. spojené komponenty, se kterými pak už pracuje OCR algoritmus, který se tyto spojené komponenty snaží klasifikovat a určit, co je to za znak. Tomuto procesu se také někdy říká prahování.

Jednoduchá binarizace

Tato metoda taktéž zvaná jako globální binarizace je jedna z metod prahování. Na začátku tohoto procesu se nastaví hodnota prahu, která je stejná po celou dobu binarizace ve všech částech obrazu. Pokud je hodnota intenzity pixelu vyšší jako daný práh, převede se hodnota na jedničku. Pokud je hodnota nižší než daný práh, přepíše se na nulu.

Nevýhoda této metody je zejména špatný výsledek pro obraz, kde se výrazně mění hodnoty jasu. Pokud je určitá část obrazu zakrytá stínem, tak bude celá tato oblast po binarizaci černá.

(a) Před binarizací (b) Po binarizace

Obrázek 11: Ukázka jednoduché binarizace

Je možné sice nastavit nižší hodnotu prahu, ale tím by se zase ovlivnila binarizace na světlejší části obrazu, kde by se ve výsledku vyskytovalo větší množství šumu.

Na obrázku 11 lze vidět použití jednoduché binarizace s hodnotou prahu 127. Je vidět, že po aplikování binarizace je dokument pouze buď černý nebo bílý.

Adaptivní binarizace

Tento způsob dokáže eliminovat právě problémy s rozdílným jasem v obraze. Hodnota prahu není udávána globálně pro celý obraz, ale obraz se prochází postupně po částech a práh je pro každou část počítán zvlášť. Existují dva nejčastější způsoby počítání lokálního prahu:

1. Průměrování - Práh je spočítán z průměrné hodnoty intenzity pixelu v okolí
2. Gaussián - Práh je váženým průměrem hodnot intezity v okolí

Otsuova metoda

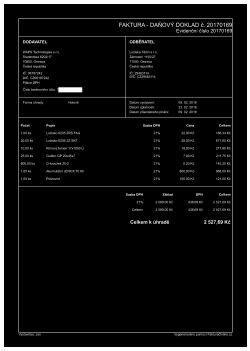
K pochopení Otsuovy metody je nutné vědět, co je to histogram obrazu. Je to v podstatě graf, který znázorňuje zastoupení jednotlivých intenzit pixelů v obraze.

Otsuova metoda je speciální adaptivní binarizací. Dá se použít v případě, kdy máme bimodální obraz (obraz, který má v histogramu dva výrazné vrcholy). V tomto případě se jako práh bere prostřední hodnota mezi těmito dvěma vrcholy.

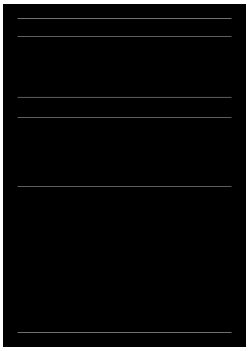
2.8 Odstranění rámečků

Dalším možným krokem v předzpracování obrazu je odstranění rámečků okolo faktury. V mé práci je však tento bod poměrně klíčový, jelikož Tesseract si v mnohých případech neumí poradit s texty, které jsou nějakým způsobem orámečkované. Pokud by to byly nepodstatné slova, které nejsou žádným způsobem pro fakturu klíčové, tak by to až tak nevadilo. Horší je však situace, kdy jsou v rámečcích důležité informace jako například celková cena faktury. Takové faktury se vyskytují celkem často.

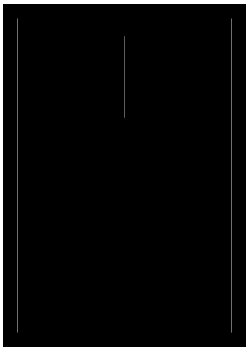
Vyzkoušel jsem několik postupů jak odstranit rovné čáry z faktury. Jedním z nich bylo řešení pomocí detekce přímek Houghovou transformací. Toto řešení však nechávalo poměrně velké



(a) Invertovaný obraz



(b) Horizontální maska



(c) Vertikální maska

(d) Výsledný invertovaný obraz bez rámečků

Obrázek 12: Ukázka metody odstraňování ráměčků

zbytky rámečků, které OCR algoritmus stále považoval za text. Nepomohlo ani následné použití filtrů a binarizace.

Nakonec jsem tento problém vyřešil pomocí morfologických operací. Nejprve se obraz prahuje s inverzí. To znamená, že černé písmena jsou bílé a bílé pozadí se stane černým. Poté si vytvořím dva strukturní elementy. Jeden pro horizontální a druhý pro vertikální přímky. Tyto elementy jsou obdélníky o velikosti 100x1 a 1x100. Pak se provádí operace otevírání pro oba druhy přímek. Tím, že se nejprve prování eroze tak se odstraní čáry, které nejsou dostatečné velké, aby byly součástí rámečku což můžou být například části písmen ve velkých nadpisech. Tímto vzniknou dva další obrazy. Jeden obsahuje pouze vertikální a druhý pouze horizontální čáry extrahované z faktury.

Poté oba tyto obrazy odečtu od invertovaného binarizovaného obrazu a provedu opět prahování s inverzí, čímž zpátky dostanu černé písmo na bílém pozadí. Následně se provede vyhlazení obrazu pomocí průměrování a další prahování, tentokrát už bez inverze. Tento poslední krok vyfiltruje poslední zbytky rámečků. Tento postup je ilustrován na obrázku 12.

3 OCR algoritmy

Optical character recognition (OCR) algoritmy slouží k přečtení veškerého textu na obrazu. V dnešní době se tyto algoritmy používají nejčastěji ke čtení dokumentů, ale například i k detekci textu v obrazu (například dopravní značky a podobně)

Vstupem do OCR algoritmu je binární obraz, který rozlišuje pozadí a ostatní objekty. V případě dokumentů je to pozadí a jednotlivá písmena. Písmena jsou reprezentována shlukem jedniček, které tvoří spojené komponenty. Spojené komponenty jsou skupiny pixelů, které jsou vzájemně propojeny.

Principem OCR algoritmu je klasifikace těchto spojených komponent. V tomto kroku je přiřazován těmto spojeným komponentám význam a to tak, že se každé komponentě přiřadí znak (písmeno, číslo nebo speciální znak). Je tedy nutné danou komponentu ohraničit a "vyřezat"z obrazu, aby zůstala jen daná komponenta, která se bude klasifikovat.

V této kapitole nejdříve porovnám různé přístupy k rozpoznávání znaků. Dále se seznámím s existujícími OCR algoritmy s otevřeným kódem, popíšu je a udělám i srovnání rychlosti a přesnosti. Vstupní obraz nebude nijak předzpracován, to znamená, že tento proces přenechám samotnému OCR algoritmu. Testovací obraz je k vidění na obrázku 13.

3.1 Různé přístupy klasifikace spojených komponent

Dnes už existuje více způsobů jak klasifikovat spojené komponenty. Dříve se používalo spíše porovnávání vzorů. V dnešní době se s rozvojem umělé inteligence vývojáři přiklání spíše k metodám založených na strojovém učení. Tyto metody využívají různé druhy různě hlubokých neuronových sítí.

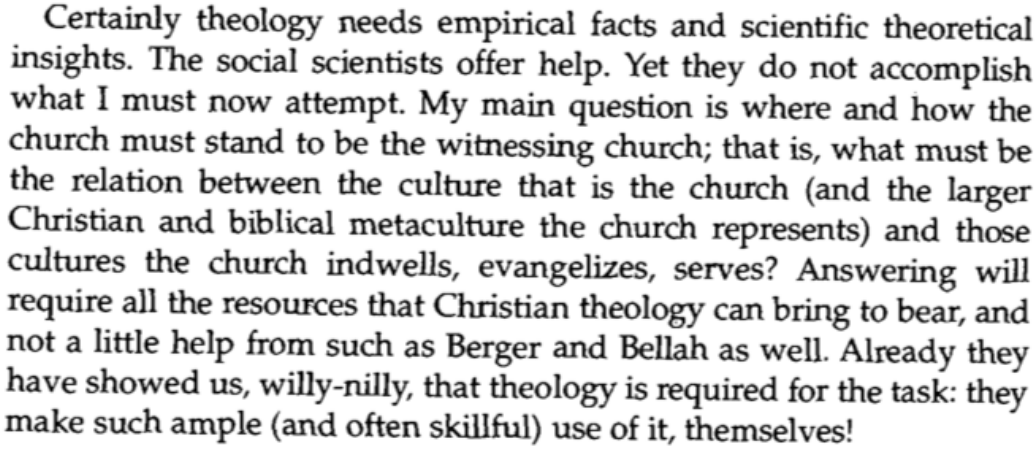
Porovnávání vzorů

Jedním ze způsobů klasifikace spojených komponent je porovnávání vzorů (pattern matching). Tento způsob funguje tak, že je dána určitá sadu znaků, která se porovnává se spojenou komponentou, která byla extrahována z obrazu. Postupně se takhle porovnává celá sadu znaků a ve výsledku se vybere ta, pro kterou byla nalezena největší shoda.

Tento přístup má několik problémů. Zaprvé je to vysoká výpočetní náročnost, jelikož je třeba projít všechny znaky datové sady, aby bylo možno určit největší shodu. Dalším problémem je fakt, že datová sada musí být poměrně obsáhlá, kvůli klasifikaci různých fontů, různých jazykových sad a podobně. Navíc tato sada musí být vždy přítomna při každé klasifikaci. Další nevýhodou je poměrně nízká přesnost[7].

Deep learning

V dnešní době je umělá inteligence poměrně populárním tématem a našla využití i ve zpracování dokumentů. V případě OCR algoritmů se využívají hlavně konvoluční neuronové sítě, které jsou

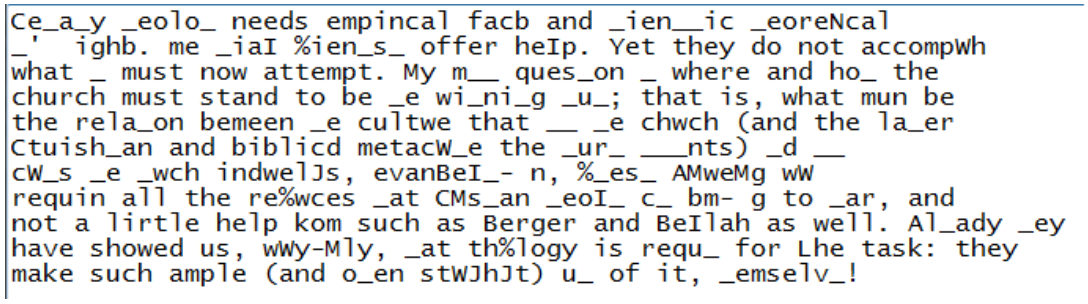


Obrázek 13: Obraz, na kterém budeme porovnávat jednotlivé algoritmy [2]

určeny pro zpracování obrazu. První fází takové sítě je trénování. To probíhá tak, že je nejprve nasbíráno větší množství obrazů znaků a jejich odpovídající znaky. Tyto data se pak postupně vkládají několikrát do sítě, která se tímto sama naučí, jaký tvar odpovídá jakému znaku. Tato síť pak například pozná, že tvar podobný kříži bude nejpravděpodobněji písmeno "T"nebo znak "+". Lepší modely využívají i sítě s buňkami LSTM. Tyto vrstvy jsou pak využívány ke klasifikaci celých slov, jelikož tyto buňky mají svou paměť a jsou vhodné pro klasifikaci sekvencí, což slova jsou.

Výhodou tohoto přístupu je jednoznačně přesnost[3]. Mnohé dnešní modely dokáží rozpoznat i ručně psaná písma s poměrně vysokou přesností[9]. Při klasifikaci už není tato metoda tak výpočetně náročná a navíc není třeba mít všechna trénovací data k dispozici při každé klasifikaci, jelikož tyto data jsou "obsaženy"v samotném modelu, který se stará o klasifikaci.

Nevýhodou je pravděpodobně čas strávený trénováním a potřeba velkého množství dat pro trénování. Aby měl model co největší přesnost, tak je třeba nasbírat velké množství fotografií písmen (klidně i v řádu desítek gigabytů). Zároveň musí být model dostatečně komplexní/hluboký, aby byl schopen poznat různě deformované znaky nebo písmena z různých fontů. S rostoucím počtem dat a složitostí sítě přímou úměrou vzrůstá i potřebný čas pro dostatečné natrénování takové sítě.



Obrázek 14: Výsledek algoritmu GOCR

3.2 GOCR

GOCR je program vytvořený Joergem Schulenburgem. Tento program je vyvíjen pod licencí GNU. Vývoj tohoto systému započal v roce 2000.

GOCR je programem, který standardně nemá grafické rozhraní. Jsou ovšem nástroje třetích stran, které tomuto programu grafické rozhraní přidají. GOCR byl napsán v jazyce C. Nevýhodou tohoto programu je omezená podpora obrazových formátů. GOCR totiž přímo nepodporuje nejběžnější formáty obrazů jako je například JPEG nebo PNG. Takové formáty musí být převedeny do formátu přenositelné mapy jako například PPM nebo PNM. GOCR je stále vyvíjen. Poslední verze programu je 0.51 a byla vydána v srpnu 2017. Oficiální stránka je k nalezení zde [5].

Testování algoritmu GOCR

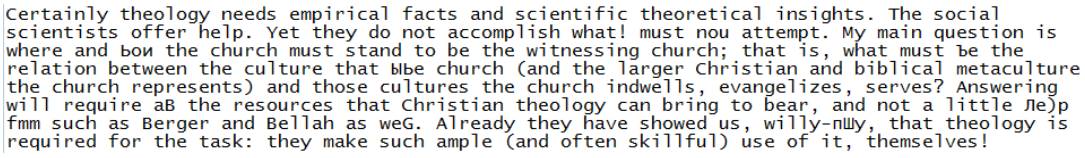
Na obrázku 14 je možno vidět, že GOCR správně přečetl 50 slov ze 122. Dá se tedy předpokládat, že GOCR implicitně neprovádí žádné předzpracování obrazu. Doba čtení byla 2,5 sekundy. Před detekcí bylo také nutno převést obrázek do podporovaného formátu.

3.3 CuneiForm

CuneiForm je OCR algoritmus vytvořený v roce 1993 ruskou firmou Cognitive Technologies původně jako komerční produkt. V roce 2007 pak firma uvedla svůj produkt jako freeware a následně v roce 2008 uvolnila své zdrojové kódy jako open-source.

CuneiForm původně také neměl své vlastní grafické rozhraní. V roce 2009 však byla vydána verze, která grafické rozhraní už obsahovala. CuneiForm je napsán v jazycích C a C++. Tento systém umožňuje zpracovat jak jednotlivé naskenované dokumenty, tak i více dokumentů najednou. CuneiForm podporuje 23 jazyků.

Nelze jednoznačně říct, jestli je tento program stále vyvíjen, jelikož jeho poslední verze vyšla v dubnu roku 2011 a je to verze s číslem 1.1. Odkaz na stažení tohoto programu je zde [6]



Obrázek 15: Výsledek algoritmu CuneiForm

Testování algoritmu CuneiForm

CuneiForm přečetl správně 111 slov ze 122. V textu lze zpozorovat výskyt písmen z jiné abecedy, což znamená, že tento OCR algoritmus provádí pouze detekci znaků nezávisle na jazyku dokumentu. Doba zpracování byla 1,76 sekund. Výsledek tohoto algoritmu vidíme na obrázku 15.

3.4 Tesseract OCR

Jako poslední jsem si nechal algoritmus Tesseract. V této podkapitole bude detailněji rozebrán, jelikož ho v mé práci používám pro čtení znaků. Oficiální stránka tohoto projektu je zde [4]

Historie

Tesseract byl původně vyvíjen jako close source projekt firmou Hewlett-Packard mezi lety 1985 až 1994. V roce 1996 byl Tesseract vydán i pro Windows. Později v roce 2005 se firma HP rozhodla vydat Tesseract jako open source projekt a od roku 2006 je tento projekt vyvíjen firmou Google.

Poslední stabilní verze tohoto programu je 3.05.01 a byla vydána 1. června roku 2017. Momentálně se pracuje na verzi 4.0, která bude používat pro čtení textů rekurentní neuronovou síť s buňkami LSTM.

Funkce

Samotný program Tesseract dokáže číst text ve více než 100 světových jazycích a podporuje formát UTF-8. Výstup z Tesseractu může mít hodně podob. Může to být soubor s prostým textem, HTML dokument nebo i PDF dokument.

Tesseract sám o sobě je program, se kterým se dá pracovat pouze v příkazové řádce, tudíž neobsahuje žádné grafické rozhraní. Existují však programy třetích stran, které tuto funkčnost doplňují. Příkladem takových programů je například OCRFeeder, VietOCR nebo QTesseract.

Při používání Tesseractu se může stát, že výsledný text nebude příliš přesný. I když Tesseract implicitně provádí určité předzpracování obrazu, tak se doporučuje udělat předzpracování ještě před tím, než obraz pošleme do Tesseractu. Mezi nejčastější techniky předzpracování dokumentace Tesseractu uvádí odšumění, binarizaci, opravu natočení, nebo odstranění okrajů naskenovaného obrazu. Na toto předzpracování existují knihovny do programovacích jazyků jako

například OpenCV nebo Leptonica, ale existují přímo i nástroje pro vylepšení kvality obrazu jako například ImageMagick.

Program Tesseract obsahuje velké množství přepínačů pro nastavení běhu programu. To zahrnuje například módy segmentace stránky, použitý OCR model, typ výstupu nebo použitý jazyk pro detekci. Publikace [11] provádí detailnější zkoumání algoritmu Tesseract.

Trénování Tesseractu

Jelikož je Tesseract založen na neuronových sítích, tak nabízí i možnost natrénování své vlastní jazykové sady. Tímto je možné vylepšit přesnost čtení, pokud má Tesseract nízkou přesnost s daty, které už byly předtrénovány. Naštěstí Tesseract obsahuje skripty, které zjednodušují a automatizují určité části trénovacího procesu.

Pro trénování Tesseractu je nutné mít data, ze kterých se bude trénovat. Nejprve se musí vytvořit textový soubor s texty, pomocí kterých se bude rozpoznávání trénovat. Tento soubor musí obsahovat všechny znaky, po kterých je požadováno, aby je Tesseract dokázal přečíst. Poté se vytisknou a naskenují stránky s textem s různými fonty a styly. Tesseract omezuje maximální počet souborů při trénování na 64. Poté se spustí skript s názvem text2image pro každý trénovací obraz.

Tento skript vygeneruje soubor s ohraničujícími obdélníky, které ohraničují jednotlivé znaky v obrazu. Obsahem tohoto souboru je dané písmeno, souřadnice levého horního rohu obdélníku a souřadnice pravého dolního rohu obdélníku.

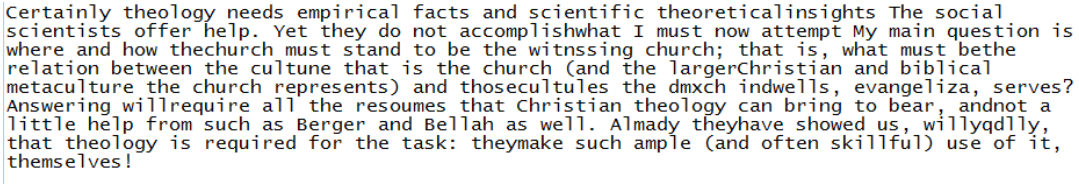
Dalším krokem je spuštění trénování pro každý vygenerovaný soubor s obdélníky zvlášť. Tento postup pro každý soubor s obdélníky vytvoří soubor s rysy každého písmene, které se Tesseract naučil. Poté se musí všechny tyto soubory zkombinovat do jednoho pomocí skriptu unicharset_extractor.

Dalším textovým souborem který je nutno vytvořit je soubor font_properties, který definuje vlastnosti fontu jako je například kurzíva nebo tučnost písma. Pro každý font tento soubor musí obsahovat řádek s názvem fontu a 0 nebo 1 pro každou vlastnost fontu.

Poté co jsou zkombinovány všechny fonty, tak se provádí shlukování. To se provádí za pomocí utilit shapeclustering, mftraining a cntraining. Tyto utility vytvoří soubory shapetable, inttemp, pffmtable a normproto.

Nyní se musí vytvořit soubor unicharambigs, který popisuje podobnosti mezi znaky nebo skupinou znaků. Nejčastější podobnost je mezi znakem m a znaky r a n.

Tímto je proces trénování u konce. Poslední věc kterou je třeba udělat, je dodat předponu *lang.* pro soubory shapetable, normproto, intemp, pffmtable a unicharset a spustit program s názvem combine_tessdata, který vytvoří jazykový soubor, který pak lze použít při čtení dokumentu.



Obrázek 16: Výsledek algoritmu Tesseract

Testování algoritmu Tesseract

Výsledek tohoto algoritmu vidíme na obrázku 16. Tesseract správně přečetl 114 ze 122 slov. Doba čtení byla 9,5 sekundy. I přes jeho pomalé zpracování je nejpřesnější. To je také jeden z důvodů, proč jsem si ho vybral pro vyřešení mého problému čtení faktur.

3.5 Srovnání OCR algoritmů

V tabulce 1 je uvedeno srovnání přesnosti a rychlostí OCR algoritmů, které jsem popsal výše. Rychlost algoritmu CuneiForm se vztahuje pouze na samotné přečtení textu. Tato rychlost nezahrnuje spuštění grafického rozhraní a načtení obrazu.

Porovnávání jednotlivých algoritmů bylo prováděno na laptopu Lenovo E550. Tento počítač má dvoujádrový procesor Intel i3 4005U s taktem 1,7 GHz. Dále má operační paměť typu DDR3 o velikosti 4GB o frekvenci 800 MHz, a integrovanou grafickou kartu.

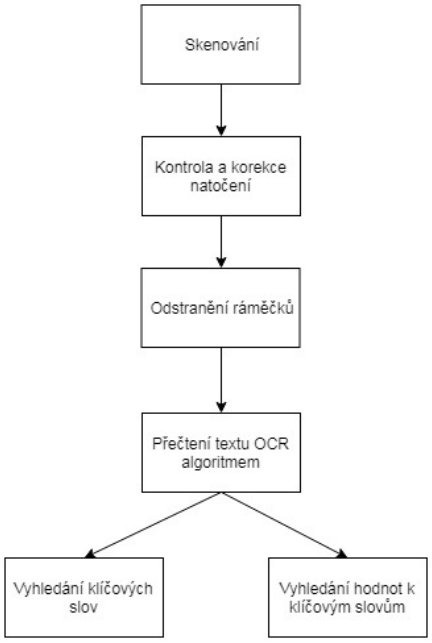
	Rychlost	Přesnost
GOCR	2,5 sekund	40,98%
CuneiForm	1,76 sekund	90,98%
Tesseract	9,5 sekund	93,44%

Tabulka 1: Srovnání OCR algoritmů

Rozsáhlejší porovnání různých OCR algoritmů a nástrojů lze najít například v publikaci [10]. Porovnání pár dalších OCR programů je také v kapitole 2 v publikaci [9].

4 Implementace řešení

V této kapitole popíšu můj postup při řešení zadaného úkolu. Nejprve zde popíšu, jaké nástroje jsem při řešení použil a následně vysvětlím jak jsem postupoval při vývoji mého programu. Na obrázku 17 je diagram mého řešení, ze kterého lze vidět, jak navržený program funguje. Před vstupem obrazu do OCR algoritmu byly použity metody předzpracování obrazu, které byly popsány v kapitole 2.



Obrázek 17: Vývojový diagram mého řešení

4.1 Použité nástroje

Pro vyřešení tohoto úkolu jsem si vybral programovací jazyk python, jelikož je typově neorientovaný a má jednoduchý a rozsáhlý balíčkovací systém. Také má nabízí podporu několika nástrojů pro zpracování obrazu. V mém řešení jsem použil knihovnu OpenCV s rozhraním pro jazyk python a knihovnu Numpy pro matematické operace. Program také obsahuje nástroj pdf2image, který dokáže převést elektronickou fakturu z PDF do obrazu pro další zpracování.

Jako OCR algoritmus jsem si vybral Tesseract, jelikož umožňuje volbu jazyka pro přesnější čtení a existuje i několik knihoven v pythonu, které nabízí rozhraní pro práci s nástrojem Tesseract. Já si pro tento účel vybral knihovnu PyOCR, která nabízí rozhraní kromě Tesseractu i pro jiné OCR programy.

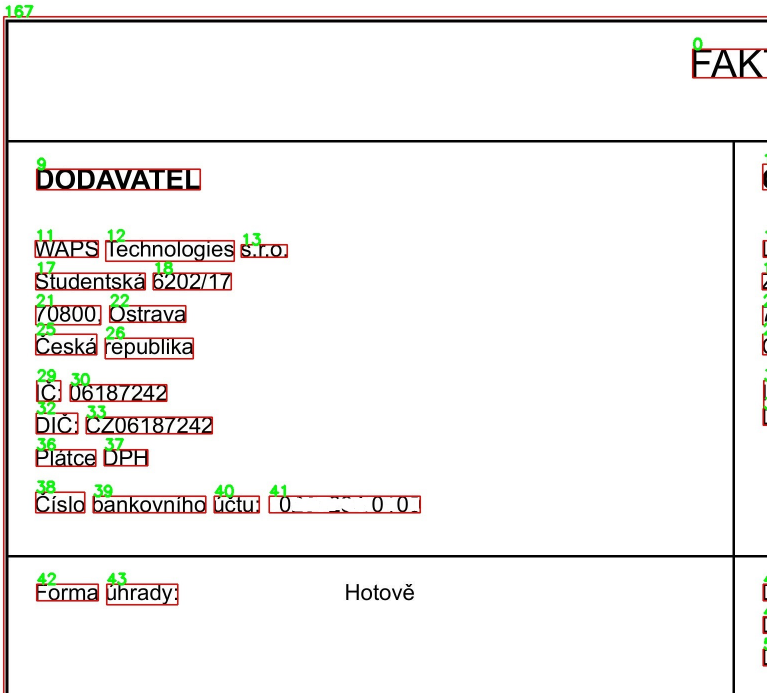
4.2 Použité postupy

Vstupem do programu je faktura buď ve formě obrazu nebo dokumentu PDF. Nejprve se provádí korekce otočení dokumentu metodou popsanou v kapitole 2.5. Poté následuje odstranění rámečků z faktury. Důvodem odstranění těchto rámečků je to, že Tesseract z nějakého důvodu někdy nedokáže dobře zpracovat text, který je v rámečcích. Buďto takový text nepřečte vůbec, nebo jako text považuje i samotný rámeček, takže přečtená hodnota OCR algoritmem nesouhlasí se skutečnou hodnotou. Důkaz tohoto je možno vidět na obrázcích 18 a 19. Pokud byly ponechány rámečky tak jak jsou, slovo "hotově"nebylo přečteno vůbec. Lze si také všimnout, že vnější rámeček stránky byl považován za text a byl ohraničen. Pokud však byly rámečky odstraněny, tak bylo slovo bez problému detekováno a správně přečteno.

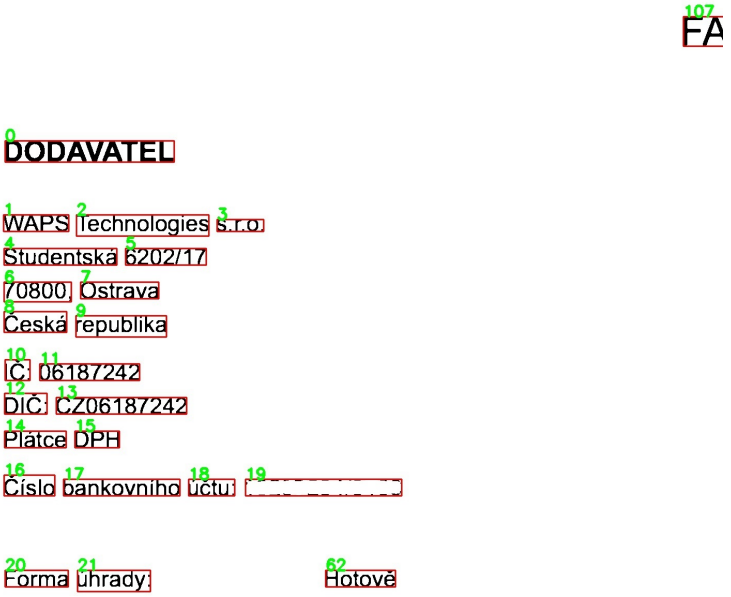
Poté přichází na řadu přečtení veškerého textu OCR algoritmem, jehož výsledkem jsou slova i s jejich ohraničujícím rámečkem.

Poslední část mého programu se stará o vyhledání klíčových informací. V programu jsou pole, které obsahují klíčová slova, které se snažíme najít. Poté se prochází všechna slova detekována OCR algoritmem a kontroluje se, zda nalezené slovo není klíčovým slovem. Pokud se klíčové slovo skládá z více slov, tak se hledá další část a to tak, že se dalším cyklem prochází opět všechna slova za detekovaným slovem a hledají se další slova, která leží napravo od prvního slova nebo pod ním.

Po vyhledání všech klíčových slov se jim přiřazují jejich hodnoty, které se opět hledají napravo od nich nebo po nimi. Pro další kontrolu se používají další mechanismy, které kontrolují, zda obsah odpovídá formátu hodnoty. Například zda text obsahuje číslo, nebo je text dostatečně dlouhý.



Obrázek 18: Ukázka fungování Tesseractu s rámečkem



Obrázek 19: Ukázka fungování Tesseractu bez rámečku

5 Testování

V této kapitole popíšu, jak mé řešení funguje na reálných dokumentech. Nejprve jsem testoval elektronickou fakturu, která byla extrahována přímo ze souboru PDF. Na obrázku 20 je k vidění jak faktura vypadá před zpracováním. Jelikož faktura nebyla skenována, tak nelze poznat korekci otočení dokumentu. Na obrázku 21 pak lze vidět fakturu už po zpracování mým programem. Faktura byla zbavena všech okrajů a byly přečteny úplně všechny slova OCR algoritmem.

Poté jsem také zkoušel, jak můj systém funguje na klasických naskenovaných fakturách. Na obrázku 22 je faktura, která je na vstupu mého programu. Lze si všimnout, že kvalita inkoustu je výrazně horší než na elektronické faktuře. Za to z velké části může proces digitalizace, který může kvalitu dokumentu poněkud zhoršit. Na obrázku 23 je však vidět, že si i s touto kvalitou můj program dokáže poradit a dokáže správně detekovat veškerý text na stránce.

Bylo třeba také otestovat funkčnost korekce otočení dokumentu, jelikož jsem vyzkoušel několik přístupů, které někdy fungovaly a někdy ne. Nakonec jsem přišel na řešení, které funguje u většiny faktur. Pro demonstraci tohoto postupu jsem uměle otočil elektronickou fakturu. Výsledek tohoto postupu můžete vidět na obrázku 24.

Zkoušel jsem i více extrémní úhly natočení faktury a i to můj program zvládl napravit. Na obrázku 25 je k vidění otočená faktura na druhou stranu než byla předchozí a o výrazně větší úhel. Jde vidět, že kousek faktury chybí. To je v pořádku pokud na chybějící části není nic důležitého.

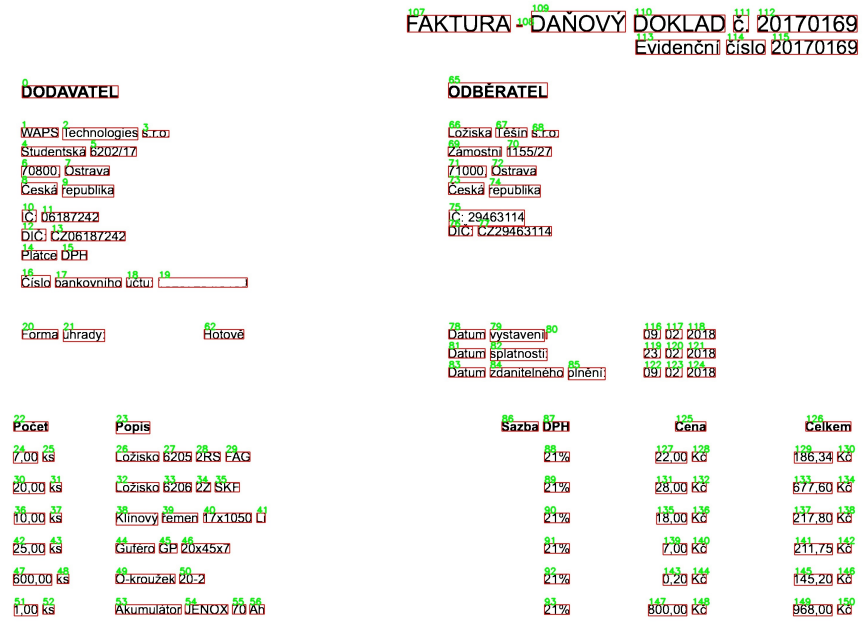
Mé řešení dokáže číst následující položky: IČO a DIČ dodavatele, IČO a DIČ odběratele, číslo faktury, datum splatnosti, datum vystavení a celkovou částku faktury k uhrazení. Další položky k vyhledání lze doplnit jednoduše přidáním dalšího pole s klíčovými slovy a funkce k vyhledání hodnoty ke klíčovému slovu. Na obrázku 26 je ukázka výstupu z elektronické faktury. Lze si povšimnout, že program nedetekoval druhé DIČ. Na obrázku 27 je ukázán výstup z tištěné faktury. Zde je vidět, že program správně vyhodnotil, že IČO a DIČ odběratele na faktuře není uveden a tak ho program nemohl najít.

FAKTURA - DAŇOVÝ DOKLAD č. 20170169
Evidenční číslo 20170169

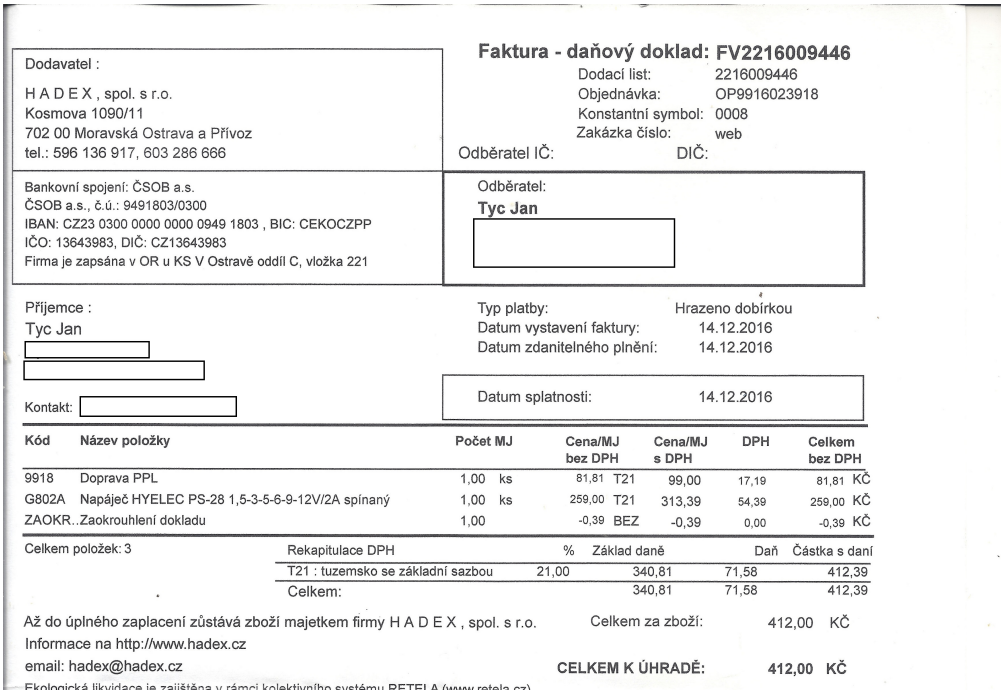
DODAVATEL	ODBĚRATEL
WAPS Technologies s.r.o. Studentská 6202/17 70800, Ostrava Česká republika IČ: 06187242 DIČ: CZ06187242 Plátce DPH Číslo bankovního účtu:	Ložiska Těšín s.r.o. Zámostní 1155/27 71000, Ostrava Česká republika IČ: 29463114 DIČ: CZ29463114
Forma úhrady: Hotově	Datum vystavení: 09. 02. 2018 Datum splatnosti: 23. 02. 2018 Datum zdanitelného plnění: 09. 02. 2018

Počet	Popis	Sazba DPH	Cena	Celkem
7,00 ks	Ložisko 6205 2RS FAG	21%	22,00 Kč	186,34 Kč
20,00 ks	Ložisko 6206 2Z SKF	21%	28,00 Kč	677,60 Kč
10,00 ks	Klínový řemen 17x1050 Li	21%	18,00 Kč	217,80 Kč
25,00 ks	Guféro GP 20x45x7	21%	7,00 Kč	211,75 Kč
600,00 ks	O-kroužek 20-2	21%	0,20 Kč	145,20 Kč
1,00 ks	Akumulátor JENOX 70 Ah	21%	800,00 Kč	968,00 Kč

Obrázek 20: Elektronická faktura před zpracováním



Obrázek 21: Elektronická faktura po zpracování



Dodavatel :

H A D E X , spol. s r.o.
Kosmova 1090/11
702 00 Moravská Ostrava a Přívoz
tel.: 596 136 917, 603 286 666

Bankovní spojení: ČSOB a.s.
ČSOB a.s., č.ú.: 9491803/0300
IBAN: CZ23 0300 0000 0000 0949 1803 , BIC: CEKOCZPP
IČO: 13643983, DIČ: CZ13643983
Firma je zapsána v OR u KS V Ostravě oddíl C, vložka 221

Příjemce :
Tyc Jan

Kontakt:

Faktura - daňový doklad: FV2216009446

Dodací list: 2216009446
Objednávka: OP9916023918
Konstantní symbol: 0008
Zakázka číslo: web
Odběratel IČ: DIČ:

Odběratel:
Tyc Jan

Typ platby: Hrazeno dobírkou
Datum vystavení faktury: 14.12.2016
Datum zdanitelného plnění: 14.12.2016

Datum splatnosti: 14.12.2016

Kód	Název položky	Počet MJ	Cena/MJ bez DPH	Cena/MJ s DPH	DPH	Celkem bez DPH
9918	Doprava PPL	1,00 ks	81,81 T21	99,00	17,19	81,81 KČ
G802A	Napáječ HYELEC PS-28 1,5-3-5-6-9-12V/2A spínaný	1,00 ks	259,00 T21	313,39	54,39	259,00 KČ
ZAOKR.	Zaokrouhlení dokladu	1,00	-0,39 BEZ	-0,39	0,00	-0,39 KČ

Celkem položek: 3

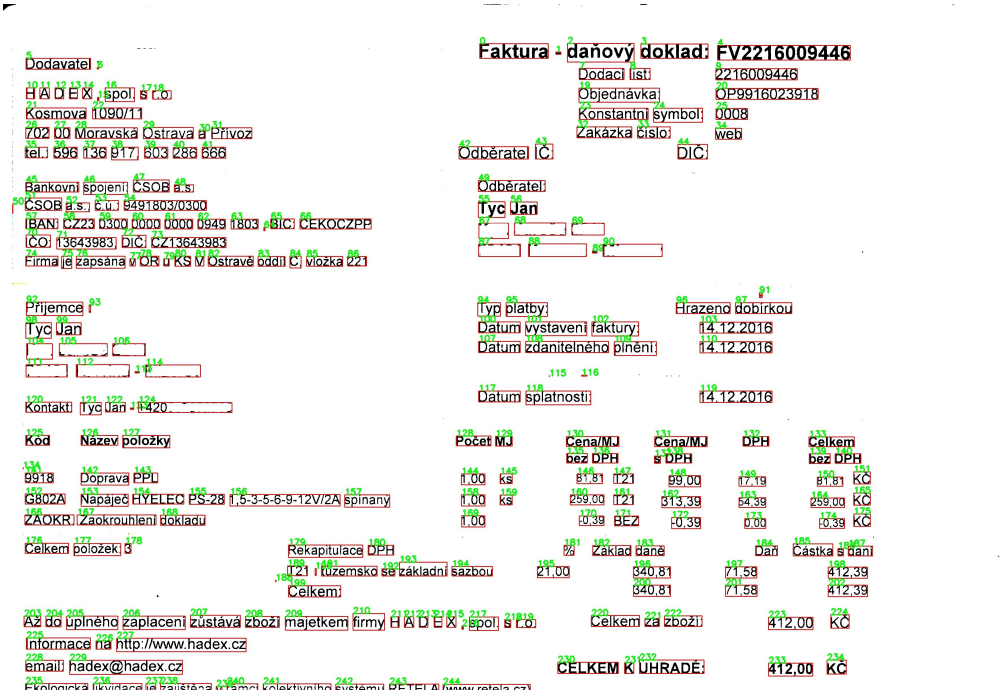
Rekapitulace DPH	%	Základ daně	Daň	Částka s daní
T21 : tuzemsko se základní sazbou	21,00	340,81	71,58	412,39
Celkem:		340,81	71,58	412,39

Až do úplného zaplacení zůstává zboží majetkem firmy H A D E X , spol. s r.o.
Informace na http://www.hadex.cz
email: hadex@hadex.cz
Ekologická likvidace je zajištěna v rámci kolektivního systému RETELA (www.retela.cz)

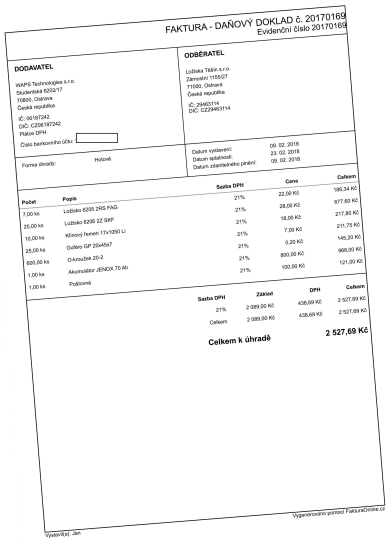
Celkem za zboží: 412,00 KČ

CELKEM K ÚHRADĚ: 412,00 KČ

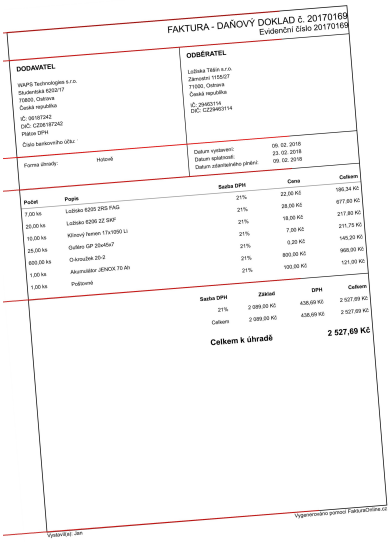
Obrázek 22: Tištěná faktura před zpracováním



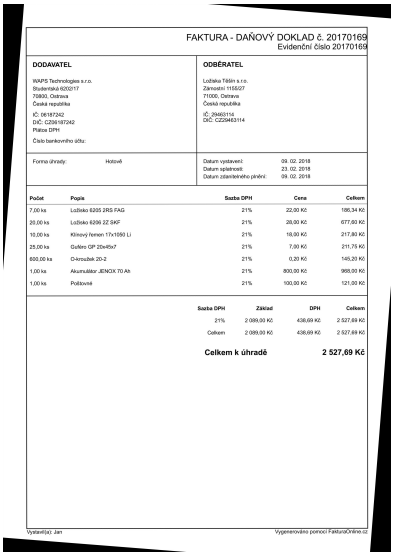
Obrázek 23: Tištěná faktura po zpracování



(a) Uměle otočená faktura

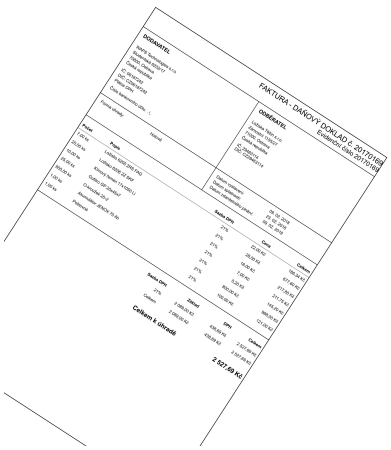


(b) Detekované vodorovné přímky, pomocí kterých určuji úhel natočení

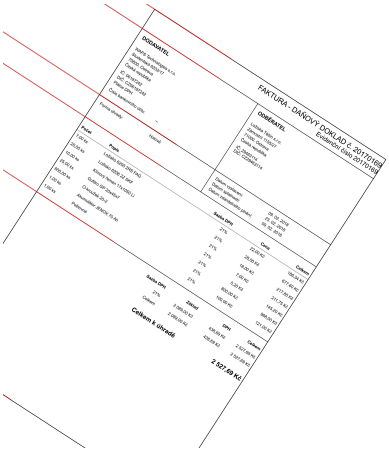


(c) Správně otočená faktura

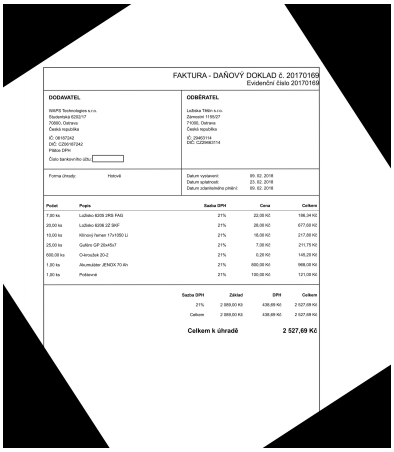
Obrázek 24: Ukázka korekce natočení faktury



(a) Uměle otočená faktura

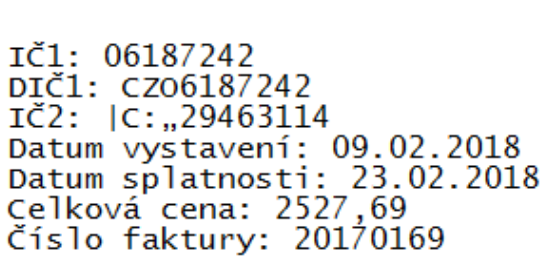


(b) Detekované vodorovné přímky, pomocí kterých určuji úhel natočení

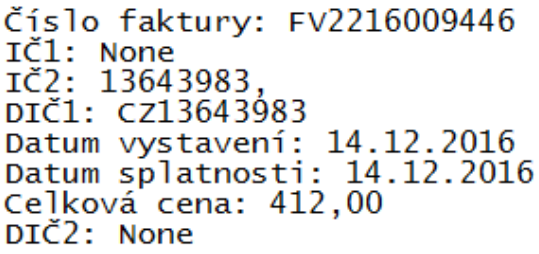


(c) Správně otočená faktura

Obrázek 25: Ukázka extrémní korekce natočení faktury



(a) Výstup z elektronické faktury



(b) Výstup z tištěné faktury

Obrázek 26: Ukázka výstupu faktur

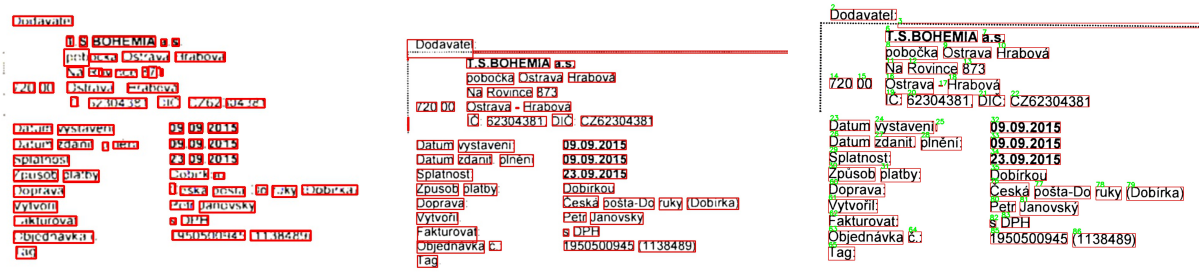
Předmětem testování bylo i měření doby částí i celku programu. Stroj na kterém bylo mé řešení měřeno je stejný jako laptop, na kterém se testovala rychlost OCR algoritmů v kapitole 3.5. Výsledek tohoto testování popisuje tabulka 2.

	Doba běhu (s)
Převod PDF na obraz	5,032
Detekce a korekce natočení	1,447
Odstranění ráměčků	4,722
Čtení OCR algoritmem	15,526
Uložení OCR hodnot	0,025
Vyhledání hodnot	0,013
Celkový čas běhu programu	28,130

Tabulka 2: Měření doby běhu programu

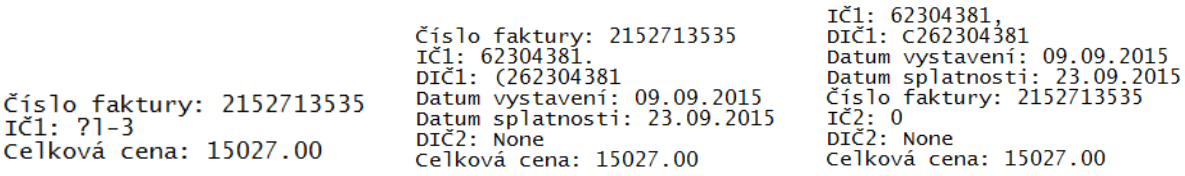
Z tabulky lze zjistit, že nejnáročnější proces co se času týče je čtení obsahu dokumentu OCR algoritmem a předzpracování obrazu. Samotná extrakce důležitých informací není nijak výpočetně náročná.

Dále jsem také experimentoval s kvalitou naskenované faktury. Zkoušel jsem nastavení skeneru v hladinách 200, 300 a 600 DPI abych zjistil, jak vysoká musí být kvalita naskenovaného obrázku, aby bylo možno přečíst co nejvíce podstatných věcí. Zkoumal jsem také vliv kvality obrázku na dobu běhu programu. Nejprve zde porovnám detekci jednotlivých slov podle různé kvality. Na obrázku 27 je srovnání jednotlivých hodnot DPI. Lze si všimnout, že hodnota 200 DPI je příliš nízká pro kvalitní přečtení. Můj program totiž v kroku odstraňování rámečků poškodil hodně textu, který sice byl detekován, ale nebyl správně přečten. Lépe dopadla faktura s 300 DPI. Zde už je kvalita textu o dost lepší a i úspěšnost čtení se zvýšila. Nejlépe samozřejmě dopadl dokument naskenován s hodnotou 600 DPI. Zde jsou písmena ostré a OCR algoritmus neměl žádný problém s čtením. Obrázek 28 ukazuje co všechno byl můj program schopný detekovat a přečíst. Opět se zde projevuje, že čím vyšší je DPI, tím lepší je výsledek mého programu.



(a) Faktura 200 DPI (b) Faktura 300 DPI (c) Faktura 600 dpi

Obrázek 27: Srovnání detekce podle DPI



(a) Faktura 200 DPI (b) Faktura 300 DPI (c) Faktura 600 dpi

Obrázek 28: Srovnání výsledků mého programu podle DPI

V tabulce 3 je uvedeno srovnání doby běhu mého programu v závislosti na velikosti DPI. Hodnoty v tabulce jsou uváděny v sekundách. Zde se logicky opět projevilo, že čím vyšší je hodnota DPI, tím déle trvá zpracování faktury, jelikož počet pixelů narůstá společně s hodnotou DPI. V tomto testu nebyl prováděn převod z PDF dokumentu, jelikož byly faktury skenované rovnou jako obraz.

	200 DPI	300 DPI	600 DPI
Korekce natočení	0,601	1,435	3,829
Odstranění rámečků	1,156	2,762	11,051
Čtení OCR algoritmem	28,144	40,456	42,608
Uložení hodnot z OCR	0,024	0,032	0038
Vyhledání hodnot	0,01	0,017	0,019
Celkový čas běhu programu	30,95	46,07	60,43

Tabulka 3: Měření doby běhu programu v závislosti na DPI

6 Závěr

Zpracování dokumentů je oblast, která dokáže výrazně usnadnit práci v mnoha oblastech. Můj úkol se mi podařilo zdárně vyřešit, ale je zde mnoho oblastí, které by se daly vylepšit. Jednou takovou oblastí je detekce položek faktury, což by pomohlo zvýšit automatizaci v některých firmách a to tím, že by se detekované položky automaticky zaúčtovaly například do účetního systému, nebo by se vložily do skladového hospodářství.

Další možnost rozšíření tohoto programu je jeho přepracování, aby dokázal fungovat obecně na jakékoliv dokumenty. Ať už to jsou účtenky, doklady nebo celé smlouvy či novinové články. K tomuto by mohl výrazně pomoct jiný způsob segmentace stránky.

Jelikož výstup z OCR někdy neodpovídá skutečným slovům, tak slovníky v mém řešení musí obsahovat slova, které OCR přečetlo se špatným znakem (například "|čo"místo "Ičo"). V tomto směru by pomohl nějaký algoritmus, který by dokázal zjistit míru podobnosti daných slov, čímž by se snížil počet obsažených slov ve slovnících klíčových slov. Jeden z možných řešení je popsán v publikaci [8] v kapitole 2.4.2.

Literatura

[1] https://en.wikipedia.org/wiki/Hough_transform#/media/File:Hough-example-result-en.png

[2] http://blog.bobkuo.com/2011/02/installing-and-using-tesseract-2-04-on-mac-os-x-10-6-6-with-homebrew/

[3] https://github.com/tesseract-ocr/tesseract/wiki/4.0-Accuracy-and-Performance

[4] https://github.com/tesseract-ocr/tesseract

[5] http://jocr.sourceforge.net/

[6] https://cognitive-openocr-cuneiform.en.softonic.com

[7] http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.661.1089&rep=rep1&type=pdf

[8] Larsson, A., SEGERÅS, T., & Segerås, T. (2016). Automated invoice handling with machine learning and OCR Automatiserad fakturahantering med maskininlarning och OCR.

[9] Springmann, Uwe & Najock, Dietmar & Morgenroth, Hermann & Schmid, Helmut & Gotscharek, Annette & Fink, Florian. (2014). OCR of historical printings of latin Texts: Problems, prospects, progress. ACM International Conference Proceeding Series. 10.1145/2595188.2595205.

[10] Vithlani, Purna & C.K.Kumbharana, C.K.Kumbharana. (2015). Comparative Study of Character Recognition Tools. International Journal of Computer Applications. 118. 31-36. 10.5120/20774-3274.

[11] Patel, C., Patel, A., Patel, D.R., Shinde, A.A., Wu, H., & Liang, J. (2012). Optical Character Recognition by Open source OCR Tool Tesseract: A Case Study.

[12] Bin Yu, Anil K. Jain, A robust and fast skew detection algorithm for generic documents, Pattern Recognition, Volume 29, Issue 10, 1996,